Optimising Fault Tolerance in Real-time Cloud Computing IaaS Environment

Bashir Mohammed[1,†], Mariam Kiran[2] ,Irfan-Ullah Awan[3] and Kabiru .M. Maiyama[4]
School of Electrical Engineering and Computer Science
University of Bradford
Bradford, BD7 1DP, UK
b.mohammed1@bradford.ac.uk[1], m.kiran@bradford.ac.uk[2], i.u.awan@bradford.ac.uk[3], k.m.maiyama@bradford.ac.uk[4]

***Abstract*— Fault tolerance is the ability of a system to respond swiftly to an unexpected failure. Failures in a cloud computing environment are normal rather than exceptional, but fault detection and system recovery in a real time cloud system is a crucial issue. To deal with this problem and to minimize the risk of failure, an optimal fault tolerance mechanism was introduced where fault tolerance was achieved using the combination of the Cloud Master, Compute nodes, Cloud load balancer, Selection mechanism and Cloud Fault handler. In this paper, we proposed an optimized fault tolerance approach where a model is designed to tolerate faults based on the reliability of each compute node (virtual machine) and can be replaced if the performance is not optimal. Preliminary test of our algorithm indicates that the rate of increase in pass rate exceeds the decrease in failure rate and it also considers forward and backward recovery using diverse software tools. Our results obtained are demonstrated through experimental validation thereby laying a foundation for a fully fault tolerant IaaS Cloud environment, which suggests a good performance of our model compared to current existing approaches.**

Keywords: Cloud computing; fault tolerance; virtual machine; pass rate; fail rate;cloud fault handler;

I. INTRODUCTION

Cloud computing has emerging as a popular paradigm and an attractive model of providing computing ,IT infrastructure, network and storage resources to end-users in both large and small enterprises [42]. It relies on sharing of resources to accomplish coherence and economies of scale, and the broader concept of shared services and converged infrastructure is at the foundation of cloud computing. Fault tolerance in cloud is the property that enables a system to continue functioning properly in the event of any failure of one or some of its components and it is of primary concern in the cloud computing environment[39]. In real time high performance large scale dynamic and complex systems in cloud, failures could happen due to varying execution environments, system components removal and addition, or intensive workload on the servers [38, 42]. Steps need to be taken in order to handle the possible failures emerging within the cloud infrastructures. Fault tolerance techniques are designed around concepts of fault tolerance principles, whereby they dictate that, even in the presence of faults the system must be able to keep working to a level of satisfaction [42]. Thus a fault-tolerant model in cloud infrastructure is needed to enable a system to continue its intended operation, possibly at a lesser degree, rather than breaking down completely. Achieving high availability in is huge challenge because the number of Virtual machines (virtual nodes) and physical servers involved in the clouds are of hundreds to thousands in number, increasing their probability and risk of failure. It is imperative to node that failures cost the cloud vendors, not only their businesses and clients, but also their reputations. To prevent these mishaps, steps are taken to handle the possible failures emerging within cloud infrastructures.

For instance in 2011, there was a Microsoft Cloud service outage which lasted for approximately 2.5 hours [30]. In the same week there was a Google Docs service outage which lasted for about an hour. This is as a result of memory leak due to a software update [42]. In Jan. 2014, one of Googles services (Gmail) was down for about 25 – 50 min, while Facebook reported an unavailable service for photos and "likes" in Oct. 2013. Additionally, in September 2012 GoDaddy experienced a long downtime which lasted for 4 hours and approximately 5 million websites were affected [30]. In October 2012, Amazon web services was reportedly down for about 6 hours [42]. Many fault tolerance strategies have been designed to reduce the effect of fault but in this paper we propose an optimized fault tolerance strategy in real time Cloud Computing Environment to increase system availability, reduce the service time and enhanced rapid and efficient recovery from faults. The Infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) are the three essential services offered by cloud computing. Each level of service handles fault tolerance at different levels of complexity. In this paper, our approach is applied to the Infrastructure as a service delivery layer, utilizing the computing hardware resources and the virtualization hypervisor that can manage virtual machine instances running on the physical server. Further components such as the load balancers, fault tolerance engine, firewalls and networking services can be utilized by cloud datacenters to help manage and regulate fault tolerance strategies in the cloud model [29, 41]. The main contribution of this paper is to develop an optimized infrastructure for IaaS cloud platforms there by showing a considerable improvement in the current research domain by optimizing the success rate of virtual computing node or virtual machines. This paper is organized as follows: Section 2 presents the related background while

section 3 discuss the proposed system architecture and model description. Section 4 presents the experimental setup and model performance comparison and finally Section 5 concludes and presents the future work.

II. RELATED WORK

Recent research that focuses on Cloud fault tolerance [32, 33, 25-28, 23-34] is more broadly covering aspects for standard real time systems [22, 25, 27, 36, 38], [16-20]. Very few works have addressed optimised fault tolerance approaches in real time cloud computing environment with focus on achieving high system availability. FT solutions are often specific to certain cloud delivery models, focusing on either high availability frameworks, using virtual nodes for fault prediction or using user defined APIs, to help optimize the cloud performance even in faulty situations. Researchers such as [12, 15, 26, 27,] focus on a reactive fault tolerance technique while [35, 14, 20, 13] analyze the performance of fault-tolerance systems with focus on high availability. Other research addresses optimization methodologies in fault situations and propose adaptive fault tolerance in real time cloud computing systems [36, 25, 11, 10, 9]. Work by Egwutuoha et al. [22] presents a Proactive Fault Tolerance approach to HPC systems in the cloud while [19, 16, 8] examine fault tolerance framework.

However, following from the various existing approaches, this paper proposed an improved model based on an optimizing tolerance approach in real time cloud applications for running virtual machines. Using techniques based on parameters being optimized, we apply a selection rate process approach, where a virtual machine or node is selected for computation based on its previous pass rate and overall task service time. The model is designed such that if the VM does not show good performance, it can be deselected from the list of available VMs. This technique does not need to have a record and playback strategy because the guarantee of successful service completion is given by the initial decisions made at deployments of the service in the VMs.

Our results have been analyzed against traditional approaches to validate how well the cloud environment repairs and manages to fulfill the service completion tasks. It has also been validated through quantitative and experimental results by simulations for testing performance for success using four use case scenarios for fault tolerance in cloud environments. The model is presented in Figure 1.

III. PROPOSED SYSTEM ARCHITECTURE

A. Model Description

We present a working model of an optimized fail-over approach in real time cloud computing environment and a mathematical relationship that represents the fault tolerance model system using the concept of virtualization and FT checkpoint-replay scheme. Our model tolerates the faults on the basis of pass rate (PR) of each virtual node's physical server.

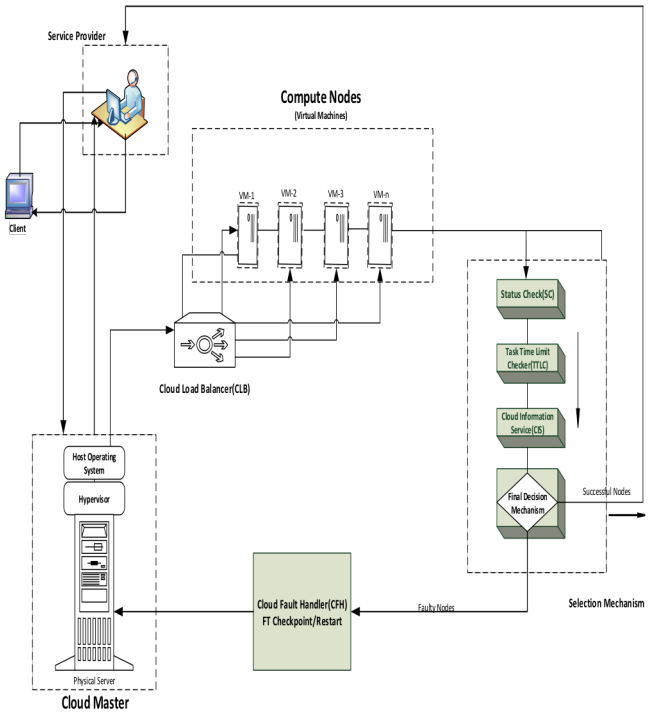


Figure 1. Proposed Optimal FT Model

$$\text{Where,} \quad \begin{cases} PR < 1, \\ PR = 1, \\ s.t \quad PR > 0, \\ 0 < PR \\ 0 < PR < 1 \end{cases} \qquad (1)$$

A virtual node is chosen for computation based on the PR of corresponding physical server and can be removed and replaced if its selected node of the server does not perform well. The checkpoint/replay model is developed using the Reward Renewal process (RRP) [5] theory with the aim of reducing the checkpoint overhead, checkpoint delay restart time and roll back time which is not covered in this paper. However, after each fault occurrence in the system, a backward recovery is performed and the VM is immediately restarted and recovered from the last successful checkpoint. Our approach has been analyzed with relation to four extreme use cases to analyze how the cloud fault handler would perform. Our model consist of four main modules namely; the service provide (SP) module, the cloud master (CM) module, the selection mechanism (SM) module and the cloud fault handler(CFH) module where the checkpoint-restart module resides.

Our fault tolerance system architecture which consists of four zones as shown in Figure 2, namely;

- The Client zone - This is where one or more client can access any service of the fault tolerance cloud data center on demand at any given time.
- The Virtualization zone - This zone is where one or more virtual machines and instances can be started up, terminated and migrated within the data center, and

also acting as a link between the Client and the fault tolerance cloud environment.

- The Fault tolerance zone - The FT zone is where the Hypervisor and Virtual machine monitor (VMM) exists supporting a high availability cloud service level and Service level objectives.
- The Hardware zone - Under this zone, we have one or more distributed data centers in different locations with each datacenter consisting of numerous physical servers which provides hardware infrastructure for starting up virtual machines instances [7].

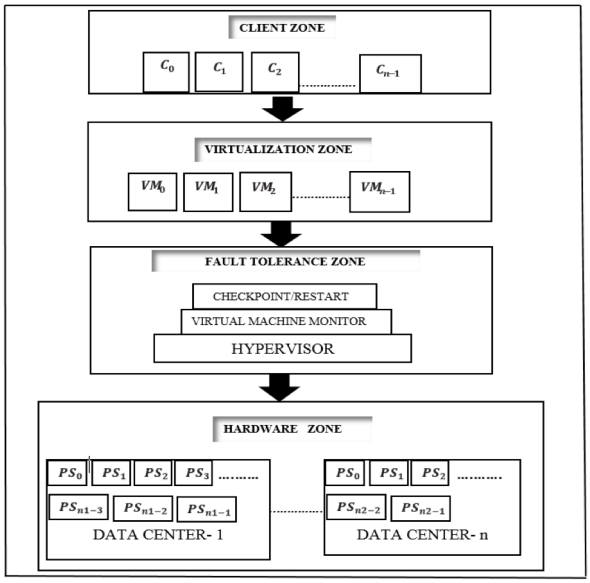


Figure 2 Proposed System Architecture

As presented in equation 2, we assume that a Fault Tolerance model (FT_m) of a cloud system be represented by a sequence of five elements (Quintuple), or finite ordered list of elements.
We define Pass Rate (PR) as the fraction or percentage of successful virtual nodes in the system after executing a complete computing cycle, and Fail Rate (FR) as the level or rate at which the virtual node of the system fails. The failure rate of the system depends on the time, status check and task time limit checker.

Table 1 Parameters of our architectural model

Parameters	Meaning
FT_m	FT model of a cloud computing system
C	A client set composed of n-users
DC	Data Centre
FT_s	Set of defined FT service levels
OBJ_f	Objective Function for optimizing a FT cloud
PR_A	Pass Rate Algorithm
FT_d	Fault Tolerance Level
$Chk(Opt)$	Checkpoint optimization strategy

$$FT_m = \left(C, DC, FT_s, OBJ_f, PR_A\right) \quad (2)$$

$$\begin{cases} OBJ_f = max(FT_L), \\ s.t \quad FT_s \in [0,1], \\ \quad\quad Chk(Opt) \end{cases} \quad (3)$$

Where PR_A is an algorithm which selects the optimal Pass Rate, $C = \{c_0, c_1, c_3, \dots, c_{n-1}\}$ represents a set composed of n-clients that may request for services separately, FT_s represents a defined set of fault tolerance service levels by the Cloud service provider, OBJ_f is the cloud fault tolerance optimizing objective function and $DC = \{dc_0, dc_1, dc_3, \dots, dc_{n-1}\}$ represents a data center set which is made up of dc_n data centers, where $dc_i = \{ps_0, ps_1, ps_3, \dots, ps_{i_{dci-1}}\}$ and ps_{i_k} $(0 \le k < dc_i)$ is the kth physical server of the ith data center dc_i.

B. Component Module Description of the Model

Figure 1 shows our proposed adaptive model. A set of nodes are created by requests from the resources of the host machine or the physical server. This is achieved by the virtual machine monitor (VMM) or a hypervisor. The host machine is the server where the VMM runs guest virtual machines. The VMM presents the guest operating systems with virtual operating platforms and also manages the execution of these guest operating systems. It also retains records of all virtual nodes created from the different host servers. In addition it retains and manages the record during the process when a cloud load balancer assigns job to a virtual node of a specific host server in order to evaluate the pass rate. The various components are:

- Service provider (SP) – is responsible for forwarding the task submitted by the client to the Cloud Controller (CC). It returns results obtained from the cloud controller to the cloud user.
- Cloud Master (CM) –is one of the key module in the cloud architecture. It has a direct link or connection to the SP. Virtualization is done here with the help of a low level program called a hypervisor which provides system resources access to virtual machines and also creates a virtual environment. In addition, it also keeps record of virtual nodes and their corresponding physical nodes each time a virtual node is created from the available resources of the physical servers. A set of virtual nodes can be created from the resources of a single physical server. The virtual node IDs, server IDs and Pass Rate are all contained in the Cloud Information Service (CIS), which helps to identify the virtual nodes and keeps record of the number of times tasks are assigned to any virtual nodes of a particular corresponding host or physical server.
- Cloud Load balancer (CLB) -The CIS is also available to the CLB and distributes the load based on information from the record of the physical systems used for virtualization. The CLB will only assign task to those virtual nodes whose corresponding physical servers are having a high pass rate.
- Cloud Fault Handler (CFH) – is a critical module in our model because when a task fails, it is allowed to be restarted from the last successful checked pointed state rather than starting from the beginning. It is an efficient task level fault tolerance technique for long running applications because it keeps the system in operation and not break down completely whenever a fault is detected[7, 12]. A scenario where a virtual node develops a fault as a result of some transient faults that occurred in the remote host server of the

corresponding virtual node or due to some recoverable temporary software faults present in the cloud controller, in these situations the CFH takes full responsibly and the cloud information service record table gets updated. However, if there is no virtual node executing under that host server, then CFH immediately restarts the remote server and the cloud load balancer is informed not to assign any task to the virtual nodes of the corresponding server. It then applies a fault detection strategy and a Checkpoint/Replay technique thereby making the virtual node of that host or physical server available for future request.

- Selection Module (SM) – The selection module is made up of the following sub modules; namely Status Check (SC), Cloud Information Service Record Table (CIS) and Task Time Limit Checker (TTLC). This module provides the crucial process and is made of the SC, TTLC, CIS record table, FT check-pointer and the final selection mechanism. Here SC checks the status of the each virtual node, and if the *Status* is *Pass* then the task deadline time is checked by the TTLC. If both SC and TTLC are pass, then pass rate (PR) of the corresponding node is increased and forwarded to the decision mechanism module for final selection process. But if both SC or TTLC *fail*, then the corresponding virtual machine is not forwarded for final selection and instead the node is forwarded to the Cloud fault manager for fault detection and recovery. In a scenario where SC is success but the task is not completed within the time limit, then the pass rate in the CIS record table of that particular node is decreased and that node is not forwarded to the final decision mechanism sub module. In addition, the final selection mechanism contains all virtual nodes that successfully passed the SC and TTLC module and after this point the node with the highest pass rate value is selected and checkpoint is made. But if all nodes failed then the CFH takes over and the Checkpoint/Replay technique is applied where a backward recovery is performed with the help of the last successful checkpoint. It should be noted that if there exist more than one node with the same PR value then a node will be selected at random. Following are the conditions and rules considered in the approach.

Table 2. Rules of the system

S/N	Rules	SC	TTLC	Outcome
1.	Condition-1	1	1	Continuous Pass
2.	Condition-2	1	0	Partial Pass
3.	Condition-3	0	1	Partial Fail
4.	Condition-4	0	0	Continuous Fail

Here in the table above "0" denotes Fail, and "1" denotes Pass

C. Checkpoint/Replay using Reward Renewal Process (RRP)

The scheme uses a checkpoint model that follows a RRP [5-7], where after each failure occurrence in the system, backward recovery is performed and the application is immediately restarted and recovered from the last successful checkpoint. In other words the fault is repaired before the last task time deadline is reached, and whenever all nodes fails, the system state is save and the application will be restarted from the last successful checkpoint as shown in Figure 3.

TABLE 3. Parameters of Fault Tolerance Model

$\boldsymbol{\Omega}_i$	The cycle between failure and failure(i+1)
$\boldsymbol{J}_n$	Time of the ith checkpoint
T_{Rol}	Roll Back Time
R_p	Restart Point
T_{Rec}	Recovery Point
T_F	Failure Point
T_{cy}	Time interval of a failure cycle
T_{OV}	Checkpoint Overhead
ΔJ	Time interval between consecutive checkpoint

Let $(T_{ov}, T_{rec}, T_{Rol})$ of each cycle be a sequence of independent identically random variable $(L_1, L_2, L_3), \ldots,$ which is dependent on any point in time failure occurs in the system Ω stands for the k^{th} time between failures in each computing cycle.

$$E\,[T_{ov}] < \infty \tag{4}$$

In order to improve the checkpoint mechanism performed by our system, we looked at how to determine checkpoint intervals that decreases the checkpoint restart time delay. Studies have shown that in order to balance the T_{ov} and roll back time of our application, checkpoint should not be carried out too regularly. So, the time delay can be expressed as

$$L_t = \sum_{i=0}^{x_t} T_i \tag{5}$$

Where, $x_t = \sup\{n: J_n \leq t\} = \max\{n \in (1,2,3\ldots.)|\, x_t \leq t\}$,

And J_n refers to the k^{th} failure time of intervals $[J_n, J_{n+1}]$ which is also called renewal intervals defined as following,

$$J_n = \sum_{i=1}^{n} T_i \tag{6}$$

Equation (5) denotes the renewal reward process where L_t is dependent on $(T_{ov}, T_{rec}, T_{Rol})$ .The renewal function is defined as the expected value of the number of failures observed up to a given time *t:*

$$f(x) = E\,[X_t] \tag{7}$$

Hence, the renewal function satisfies,

$$\lim_{t\to\infty} \frac{1}{t} f(x) = \frac{1}{E\,[\Omega_1]} \tag{8}$$

Substituting (7) into (8) gives,

$$\lim_{t\to\infty} \frac{X_t}{t} = \frac{1}{E\,[\Omega_1]} \tag{9}$$

Proving the elementary renewal theorem it is sufficient to show that for an elementary renewal theorem for renewal reward processes the reward function is given as:

$$g(x) = E\,[L_t] \tag{10}$$

The reward function thereby satisfies,

$$\lim_{t\to\infty} \frac{1}{t} g(x) = \frac{E\,[\mathrm{L}_1]}{E\,[\Omega_1]} \tag{11}$$

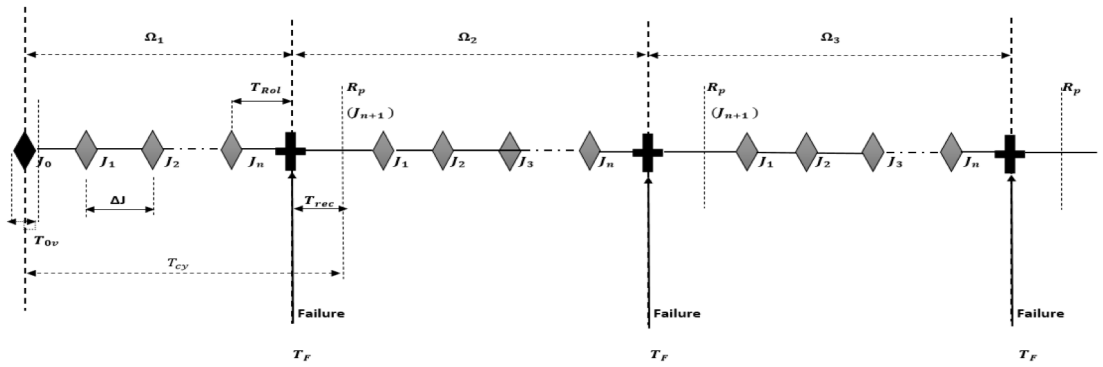


Figure 3 Checkpoint Strategy Failure Model

Algorithm 1: Cloud Controller Module Algorithm

Step 1: **Start**
Step 2: **Output** "Most Viable Node for operation with highest PR"
Step 3: **Input** "highestPassRate=1"
Step 4: **Input** PR=0.5, $\mathbf{q}_1$=1, $\mathbf{q}_2$=2.
Step 5: **If** (nodeStatus=Pass), /SC and TTLC module for that node is Pass/
{$\mathbf{q}_1 = \mathbf{q}_1+1$, $\mathbf{q}_2 = \mathbf{q}_2+1$, PR=$\mathbf{q}_1/\mathbf{q}_2$
Update CIS record table} else,
Step 6: **If** (nodeStatus=Fail), /SC or TTLC module for that node or both is Fail/
{$\mathbf{q}_2=\mathbf{q}_2+1$, PR=$\mathbf{q}_1/\mathbf{q}_2$Update CIS record table}
Step 7: **If** (PassRate>=highestPassRate)
{PassRate=highestPassRate}
Step 8: **If** (PassRate<=0)
{Inform CLB not to assign task to the node, remove the node and CC will be informed to add a new node}
Step 9: **Stop**

Substituting (10) into (11) gives

$$\lim_{t\to\infty} \frac{E\,[\mathrm{L}_t]}{t} = \frac{E\,[\mathrm{L}_1]}{E\,[\Omega_1]} \tag{12}$$

From (5), we have

$$L_t = \sum_{i=0}^{x_t} T_i \tag{13}$$

Then (12) becomes,

$$\lim_{t\to\infty} \frac{E\,[\sum_{i=0}^{x_t} T_i]}{t} = \frac{E\,[\mathbf{L_1}]}{E\,[\mathbf{\Omega_1}]} \tag{14}$$

Therefore,

$$L_t = \frac{E\,[\mathbf{L_1}]}{E\,[\mathbf{\Omega_1}]} \tag{15}$$

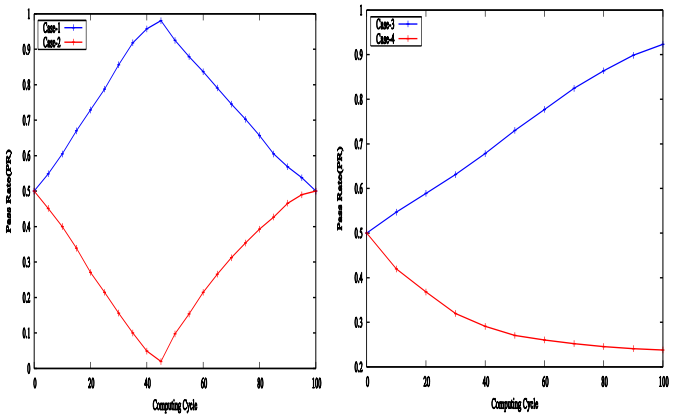
Where L_t is called the renewal reward process as derived in [5, 6]. Conversely, there is an additional time to save the system application states which is called the checkpoint overhead. To improve the checkpoint mechanism, checkpoints should not be performed too frequently in other to achieve balancing between the checkpoint overhead, recovery time and application re-computing time as derived in [2, 3, 4, 7].

Algorithm 2: Final Decision Mechanism (SM) Algorithm

Step 1: **Start**
Step 2: **Output** "Select best Node with highest PR and minimum finish time"
Step 3: **Input** from TTLC: node PassRate, q= is the number of nodes with successful SC and TTLC results.
Step 4: **Input** PR=0.5,
Step 5: **Input** highestPassRate
Step 6: **if** ($\mathbf{q}_1$==0), then {Status = fail, Conduct a backward recovery with the help of the last successful checkpoint} **else,** {Status=Pass, bestPassRate=find PassRate of the node with maximum PassRate and send outcome to the Service provide and make checkpoint}
Step 7: **Stop**

D. Use Case Scenario Analysis

We considered 100 computing cycles and present a metric analysis to evaluate the Pass and Fail scenarios of three virtual nodes (VM-1, VM-2, VM-3) respectively. Here we assumed that each VM belongs to a different host server, $\boldsymbol{P = 0.5}$, $\mathbf{q_1}$ represents the number of times virtual machine of a host server produce a Pass outcome and $\mathbf{q_2}$ represent the time the Cloud Controller's Load balancer designates a task to a virtual node.



(a) Use Case1 (Pass to Fail), Use Case 2 (Fail to Pass)
(b)Use Case 3 (Continuous Pass), Use Case 4 (Continuous Fail)

Figure 4. Use Case Scenarios for Pass and Fail

Figure 4 shows the Pass and Failure scenario comparison of our use case scenarios where the increase in PR after the required 100 computing cycles for use case-3 is 0.978, while the decrease rate (FR) for use case-4 is 0.579. This shows that the increase in PR is greater than the decrease in FR, clearly displaying a good performance of algorithm. Further scenarios for use case-1 and 2 were tested for validating the results. The obtained results indicates that use case-1 continuously passed while use case-2 continuously failed as shown in figure 4(a).While use case-3 passed for the first 100 cycles and then failed for the remaining 100 cycles figure. Use case-4 failed for the first 100 cycles and then succeeded for remaining 100cycles (Figure 4a).

IV. Experimental Setup and Performance Comparison

The experiments were conducted using CloudSim[1], where three virtual nodes were created and tested. The SFS algorithm has 100 computing cycles with every individual node executing a series of tasks at a time. While these tasks are executed in one computing cycle, every virtual node runs a diverse algorithm. The different Pass and Failure scenario obtained from this experiment displays the diversity in software and timing constraints. The selection or decision mechanism is responsible for receiving results obtained from the virtual machines before it dispatches or returns the result of the successful job to the client via the service provider.

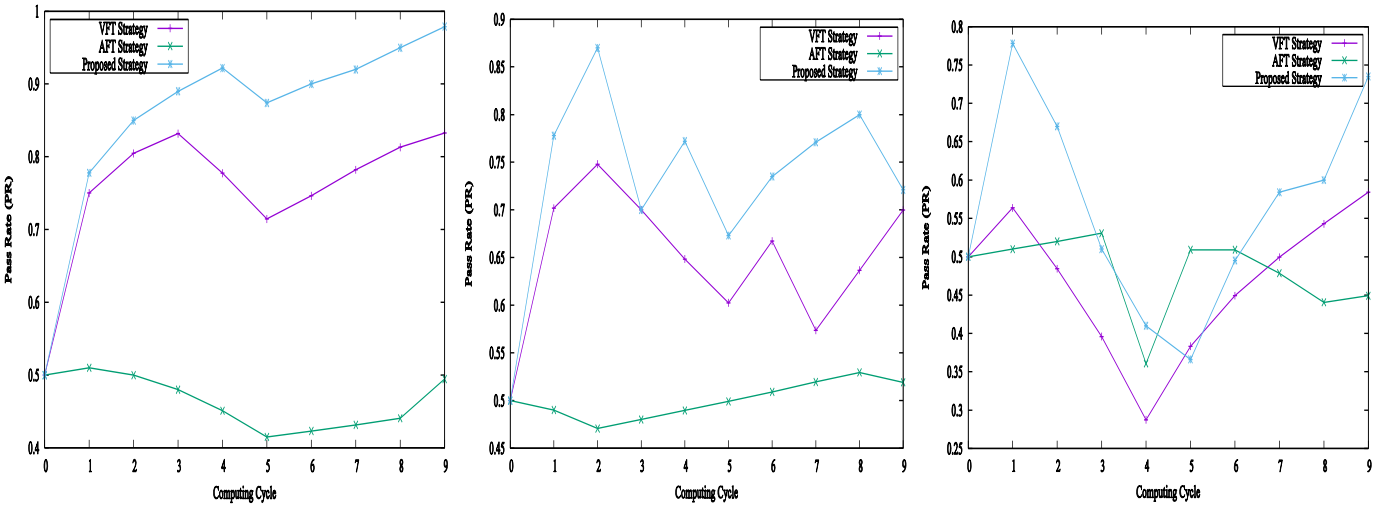
At the service provider level, the selection or decision mechanism is integrated with the Cloud controller module. In a situation where a failure occurs in one of the nodes, the system will automatically adapt a fail-over strategy and continues operating using the remaining nodes. The system will maintain and continue its operation in a steady state until all nodes have failed. A node is then selected and a checkpoint is made by the last selection mechanism to keep the status of the system for future recovery. This is done after a successful completion of one computing or instruction cycle.

We assumed that the value of x_1, x_2 PR, virtual node ID, corresponding server ID are available and that the task deadlines are given as input with initial values x_1 =1, $x_2 = 2$ and PR = 0.5 are considered for every node. Figure 4 presents results obtained from the performance comparison of our proposed strategy with other existing approaches.

A. Performance Comparison of Results

1. Virtualization and Fault Tolerance Approach (VFT) - Das et al. [12] proposed a virtualization and fault tolerance technique to reduce the service time and increase the system availability. They used a Cloud Manager module and a Decision Maker in their scheme to manage virtualization and load balancing and also handle faults. By performing virtualization and load balancing, fault tolerance was achieved by redundancy and fault handlers. Their technique was mainly designed to provide a reactive fault tolerance where the fault handler prevents the unrecoverable faulty nodes from having adverse effect.
2. Adaptive Fault Tolerance Approach(AFT) –Mailk et al. [36] proposed an adaptive fault tolerance in time cloud computing where the main essence of their proposed technique was an adaptive behaviour of the reliability weights assigned to each processing node and adding and removing of nodes on the basis of reliability.
3. Our Proposed Approach - For the purpose of evaluation, we compared our proposed strategy with the VFT approach[12] and AFT where we referred to the result obtained from our proposed strategy as the measured parameter while that of VFT and AFT are referred to as the calculated parameters respectively.

Based on the performance comparison analysis conducted from our simulated results, the standard deviation for VM-1 was estimated to be 6.5187, while that of VM-2 and VM-3 was 8.4856 and 12.004 respectively. In other words, compared to our proposed approach, this reveals that the VFT strategy deviated by 6.5187 in VM-1 as shown in figure 5(a) and by 8.4856 in VM-2 figure 5(b). A further deviation of the strategy was also noticed in VM-3 as presented in figure 5(c).This also shows the degree of discrepancy of the calculated result from the measured result. For the VFT algorithm, the increased in Pass Rate is less than our proposed strategy. Details of this model can be found in [12]. In our study, 100 computing cycles were used in simulation and assumed pass rate of 0.5 at the beginning.



(a) Virtual Machine – 1 (b) Virtual Machine – 2 (c) Virtual Machine – 3

Figure 5. Performance Comparison of VFT strategy with our Proposed Strategy

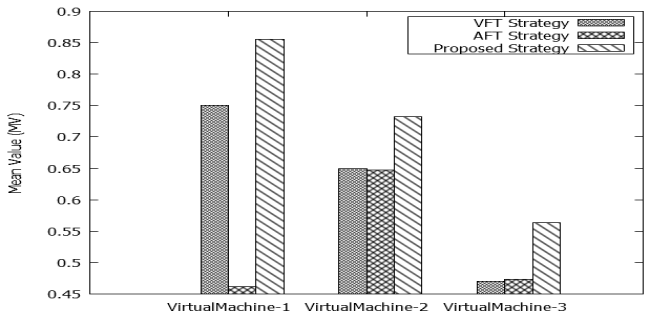


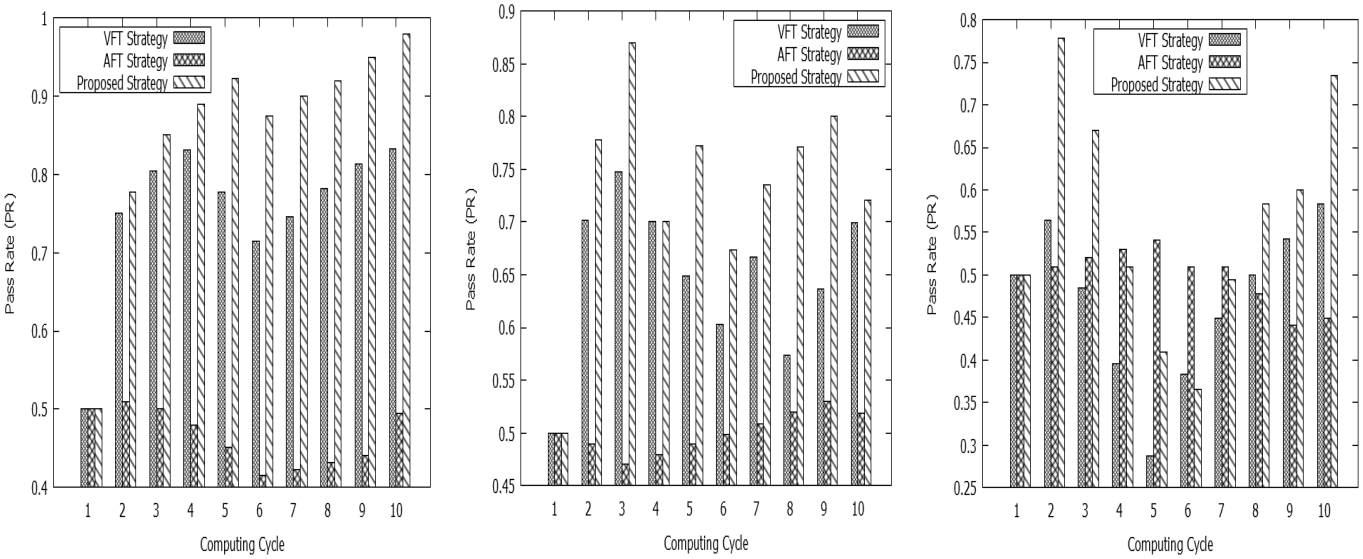
Figure 5 Mean Error Comparison

In comparing the three models, we first obtain the relative error x_{re} , then we calculated the actual error x_i which is the difference between the calculated and measure result. These are expressed as (16) and (17) respectively.

$$x_{re} = \left(\frac{q_{iCalculated} - q_{iMeasured}}{q_{iMeasured}}\right) \times 100 \qquad (16)$$

$$x_i = q_{iCalculated} - q_{iMeasured} \qquad (17)$$

Figure 5 and Figure 6 shows the mean error comparison and success rate analysis between the strategies, where we concluded that our proposed model is an improved approach. The Average percentage relative error (APE), Average absolute percentage relative error (AAPE) and Standard deviation (SD) were also obtained. Equation (18-20) gives the mathematical definition of these parameters and (Table 5) gives experimental results.



(a) (b) (c)

Figure 6. Success Rate Analysis of *VM*-1, *VM*-2 and *VM*-3

$$\varepsilon_1 = \left[\frac{1}{N}\sum_{i=1}^{N} x_{re}\right] \qquad (18)$$

$$\varepsilon_2 = \left[\frac{1}{N}\sum_{i=1}^{N} |x_{re}|\right] \qquad (19)$$

$$\varepsilon_3 = \sqrt{\frac{\sum_{i=1}^{N}(x_{re} - e_1)^2}{N-1}} \qquad (20)$$

A Performance evaluation of VFT and AFT strategy with our proposed model was carried out using x_{re}, x_i , e_1, e_2 and e_3 respectively.

TABLE 4. Parameters

Parameter	Meaning
ε_1	Total average percentage relative error
ε_2	Total average absolute percentage relative error
ε_3	Total standard deviation
x_{re}	Relative error
x_i	Actual error

TABLE 5. Performance Comparison of VFT and AFT with Proposed Strategy

		Performance Comparison Parameters		
Virtual Machine ID	Compute Cycle	ε_1(%)	ε_2(%)	ε_3(%)
VM-1	100	-11.0812	11.0812	6.5187
VM-2	100	-10.8510	10.8510	8.4856
VM-3	100	-15.6587	15.6587	12.0041

V. CONCLUSION

This paper presents a intergated virtualized failover strategy for real time computing on the cloud infrastructure. The proposed IVFS uses the pass rate of the computing virtual nodes with the fault manager using the checkpoint/replay tecnique by appying the reward renewal process thoerem. It tries to repair the fault generated before the deadline as a fault tolerance mechanism. that the scheme is highly fault tolerant because it had all the advantages of forward and backward recovery which the system takes advantage of using diverse software. Our algorithm integrates the concept of fault tolerance based on high pass rate of computing nodes and less service task finish time there increasing the system availability. With the help of the pass and fail rate analysis obtained from the experimental results as well as the performance comparison of the existing

approaches,we conclude that the proposed fault tolerance scheme gives an improved performance. In our future we will aim at addressing fault tolerance challenge especially in a large-scale high performance environment by minimizing the performance loss like checkpoint overhead, checkpoint recovery time and re-computing time.

REFERENCES

[1] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," *2010 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, pp. 446–452, 2010.

[2] D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments," *J. Supercomput.*, vol. 66, no. 1, pp. 193–228, 2013.

[3] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 700–709, 2011.

[4] R. Nassar, B. Leangsuksun, and S. Scott, "High Performance Computing Systems with Various Checkpointing Schemes 2 Full Checkpoint / Restart Model," vol. IV, no. 4, pp. 386–400, 2009.

[5] R. Gallager, "Discrete stochastic processes," no. 0, pp. 92–138, 1996

[6] G. F. Lawler, "Introduction to Stochastic Processes." p. 248, 2006.

[7] Y. Liu, R. Nassar, C. (Box) Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," *2008 IEEE Int. Symp. Parallel Distrib. Process.*, pp. 1–9, 2008.

[8] I. P. Egwutuoha, S. Chen, D. Levy, and B. Selic, "A fault tolerance framework for high performance computing in cloud," *Proc. - 12th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. CCGrid 2012*, pp. 709–710, 2012.

[9] A. Israel and d. raz, "Cost aware fault recovery in clouds," pp. 9–17, 2013.

[10] H. Yanagisawa, T. Osogami, and R. Raymond, "Dependable virtual machine allocation," *2013 Proc. IEEE INFOCOM*, pp. 629–637, 2013.

[11] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 615–626, 2012.

[12] P. Das and P. M. Khilar, "VFT: A virtualization and fault tolerance approach for cloud computing," *2013 IEEE Conf. Inf. Commun. Technol. ICT 2013*, no. Ict, pp. 473–478, 2013.

[13] M. Chtepen, F. H. a Claeys, B. Dhoedt, F. De Turck, P. Demeester, and P. a. Vanrolleghem, "Adaptive Task Checkpointing and Replication: Towards Efficient Fault-Tolerant Grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 180–190, 2008.

[14] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," *Proc. Int. Conf. Dependable Syst. Networks*, pp. 497–506, 2010.

[15] D. Singh, J. Singh, and A. Chhabra, "High availability of clouds: Failover strategies for cloud computing using integrated checkpointing algorithms," *Proc. - Int. Conf. Commun. Syst. Netw. Technol. CSNT 2012*, pp. 698–703, 2012.

[16] H. K. H. Kim, S. K. S. Kang, and H. Y. Yeom, "Node selection for a fault-tolerant streaming service on a peer-to-peer network," *2003 Int. Conf. Multimed. Expo. ICME '03. Proc. (Cat. No.03TH8698)*, vol. 2, no. 1, pp. 6–12, 2003.

[17] N. Yadav and S. K. Pandey, "Fault Tolerance In Dcdidp Using HAProxy," pp. 231–237.

[18] R. Nogueira, F. Araujo, and R. Barbosa, "CloudBFT: Elastic Byzantine Fault Tolerance," 2014

[19] E. Okorafor, "A fault-tolerant high performance cloud strategy for scientific computing," *IEEE Int. Symp. Parallel Distrib. Process. Work. Phd Forum*, pp. 1525–1532, 2011.

[20] X. Kong, J. Huang, C. Lin, and P. D. Ungsunan, "Performance, Fault-Tolerance and Scalability Analysis of Virtual Infrastructure Management System," *2009 IEEE Int. Symp. Parallel Distrib. Process. with Appl.*, pp. 282–289, 2009.

[21] S. Siva Sathya and K. Syam Babu, "Survey of fault tolerant techniques for grid," *Comput. Sci. Rev.*, vol. 4, no. 2, pp. 101–120, 2010.

[22] I. P. Egwutuoha, S. Chen, D. Levy, B. Selic, and R. Calvo, "A proactive fault tolerance approach to High Performance Computing (HPC) in the cloud," *Proc. - 2nd Int. Conf. Cloud Green Comput. 2nd Int. Conf. Soc. Comput. Its Appl. CGC/SCA 2012*, pp. 268–273, 2012.

[23] M. Amoon, "A job checkpointing system for computational grids," *Open Comput. Sci.*, vol. 3, no. 1, pp. 17–26, 2013.

[24] K. J. Naik and N. Satyanarayana, "A novel fault-tolerant task scheduling algorithm for computational grids," *2013 15th Int. Conf. Adv. Comput. Technol.*, pp. 1–6, 2013.

[25] K. Parveen, G. Raj, and K. R. Anjandeep, "A Novel High Adaptive Fault Tolerance Model in Real Time Cloud Computing," pp. 138–143 , 2014.

[26] A. Tchana, L. Broto, and D. Hagimont, "Approaches to cloud computing fault tolerance," *IEEE CITS 2012 - 2012 Int. Conf. Comput. Inf. Telecommun. Syst.*, 2012.

[27] J. Kaur and S. Kinger, "Efficient Algorithm for Fault Tolerance in Cloud Computing," *2014 IJCSIT Int. J. Comput. Sci. Inf. Technol.*, vol. 5, pp. 6278–6281, 2014.

[28] A. Ganesh, M. Sandhya, and S. Shankar, "A study on fault tolerance methods in Cloud Computing," *2014 IEEE Int. Adv. Comput. Conf.*, pp. 844–849, 2014.

[29] C.-T. Yang, Y.-T. Liu, J.-C. Liu, C.-L. Chuang, and F.-C. Jiang, "Implementation of a Cloud IaaS with Dynamic Resource Allocation Method Using OpenStack," *2013 Int. Conf. Parallel Distrib. Comput. Appl. Technol.*, pp. 71–78, Dec. 2013.

[30] ITProPortal, "ITProPortal.com: 24/7 Tech Commentary & Analysis," 2012. [Online]. Available: http://www.itproportal.com/. [Accessed: 24-Jun-2015].

[31] A. Greenberg, J. Hamilton, D. a Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2008.

[32] M. Pradesh, "A Survey On Various Fault Tolerant Approaches For Cloud Environment During Load Balancing," vol. 4, no. 6, pp. 25–34, 2014.

[33] D. Sun, G. Chang, C. Miao, and X. Wang, "Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments," *Journal of Supercomputing*, vol. 66, no. 1. J Suercomputer (), pp. 193–228, 2013.

[34] B. Mohammed and M. Kiran, "Analysis of Cloud Test Beds Using OpenSource Solutions," *2015 3rd Int. Conf. Futur. Internet Things Cloud*, pp. 195–203, 2015.

[35] S. Shen, A. Iosup, A. Israel, W. Cirne, D. Raz, and D. Epema, "An Availability-on-Demand Mechanism for Datacenters," *2015 15th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput.*, pp. 495–504, 2015.

[36] S. Malik and F. Huet, "Adaptive Fault Tolerance in Real Time Cloud Computing," *2011 IEEE World Congr. Serv.*, pp. 280–287, Jul. 2011.

[37] A. Bala and I. Chana, "Fault Tolerance- Challenges , Techniques and Implementation in Cloud Computing," *Int. J. Comput. Sci.*, vol. 9, no. 1, pp. 288–293, 2012.

[38] R. Jhawar, V. Piuri, and I. Universit, "Fault Tolerance Management in IaaS Clouds," *2012 IEEE First AESS Eur. Conf. Satell. Telecommun.*, pp. 1–6, 2012.

[39] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. N. Alsaleh, "TeachCloud : A Cloud Computing Educational Toolkit," no. 2012, pp. 1–16.

[40] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," *Natl. Inst. Stand. Technol. Spec. Publ. 800-145 7 pages*, 2011.

[41] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Cloud computing migration and IT resources rationalization," *2014 Int. Conf. Multimed. Comput. Syst.*, pp. 1164–1168, Apr. 2014.

[42] K. Bilal, O. Khalid, S. Ur, R. Malik, M. Usman, and S. Khan, "Fault Tolerance in the Cloud," no. ITProPortal, pp. 1–13, 2012.